Towards Peer-to-Peer Content Indexing

Carlos Baquero and Nuno Lopes
{cbm,nuno.lopes}@di.uminho.pt

Departamento de Informática, Universidade do Minho

Abstract

Distributed Hash Tables are the core technology on a significant share of system designs for Peer-to-Peer information sharing. Typically, a location mechanism is provided and object identifiers act as keys in the index of object locations. When introducing a search mechanism, where single words are used as keys, the key image cardinality will be driven by the word popularity and most of the present designs will be unable to load balance the index among the nodes. We present two contributions: A design that allows participating nodes to load balance the indexing of popular keys and avoid content hot-spots on single nodes; A distributed mechanism for probabilistic filtering of popular keys (with low search relevance) that paves the way for scalable full content indexing.

1 Introduction

Peer-to-Peer systems (P2P), where peers of hosts in a network share information and resources, are growing into a powerful new paradigm for decentralized distributed computing. Early system models, like those inherent to Napster [11] and Gnutella [7], have respectively shown the vulnerability of centralized indexing and the non-scalability of global query dissemination.

Building on these deficiencies, some researchers described proposals to soften the problems of global query dissemination [17, 1], keeping the possibility of key matching at query time, while others devised efficient solutions to the provision of name-based object location [16, 14, 12, 18, 10], after assuming stable key-value pairs. The later class of approaches provides scalable solutions to the insertion and querying of key-value associations, both on the number of exchanged messages and, to some extent, on the required state on each peer.

Consider a distributed index used to locate objects in a system scaling up to millions of nodes. The existing approaches only provide adequate solutions to the mapping of object identifiers into a small set of object locations. Typically, the concrete mapping is from file names (e.g. `Movie_Trailer.mpg`) into a set of IP and port addresses together with a local index number (e.g. {`411@193.136.20.1:451`, `45@193.100.34.1:325`}). In these mappings, the number of file locations can be assumed to be small, since each file name is intended to describe a concrete object and, consequently, its image can only grow as the object is physically replicated and spontaneously announced from distinct hosts.

One consensual scalability criteria is that the indexing load must be adequately distributed among peers so that one peer does not have to support an unmanageable amount of load, both in terms of required storage and communication resources. In this article we will show that existing solutions only meet this criteria by assuming a small number of values for each key, and that they will not trivially adapt to settings with very large sets of associated values (which we will refer to as the *key-image*). Such is the case when enabling search from sets of descriptive words, in which the index would map individual words instead of object identifiers as keys. In fact, except for very specific uses, such as in Freenet [4] where the object identifier must be publicized outside the system for confidentiality purposes, word search is a basic requirement for a usable document indexing system [9]. Since words typically follow a Zipf distribution [6, 2] (depicted in the line on Figures 2 and 3), with a small set of words present in many description phrases (e.g. `two`, `the`, `mp3`), it is easy to conclude how the key-image size would be affected for those words.

In the following sections we will exemplify the problems of existing solutions and describe an approach that overcomes it while keeping the design scalable. We conclude by analyzing and proposing a solution to the new problems that are introduced when full content indexing is sought.

2 Co-Domain Sensitivity

Scalable object indexing systems, providing the functionality of a distributed hash table (DHT) in a P2P setting, are mostly based on a routing strategy that progressively directs insertions and queries to the appropriate logical sections of the P2P overlay network. In order to randomize the placement of keys, the usual strategy is to map object identifiers into bit sequences, by way of a standard hash function. Some approaches [10, 16, 14] also hash the node identity, so as to establish associations between the keys and the hosts that will store them.

These techniques ensure that two keys that are lexically close will, most probably, not be so after being hashed, thus fostering a good balancing of the storage load on the overlay network. This balancing, however, only applies to the distribution of keys and not to the balancing of their associated values (key-image). One can expect that very popular keys will be randomly distributed across the network, but even a single one of those keys can still be a burden for the host that is required to store it.

This will be less likely to occur when indexing complete filenames, but when indexing single words it is easy to imagine that keys such as `mp3`, that can be found on most node contents, will force the host that stores them to keep a key-image that will grow linearly, $O(n)$, with the number n of nodes in the system.

Since for object identifier keys the same can only be expected exceptionally for very popular contents, that are stored across several nodes, it is easy to overlook this problem when designing DHT algorithms.

Our analysis of existing DHT systems, namely Chord [16], Pastry [14], CAN [12], Tapestry [18] and Kademlia [10] lead us to conclude that they are all subject to some form of co-domain sensitivity and consequently do not adapt to single word indexing.

CFS [5] offers a load balancing scheme for P2P storage of large files. Files are mapped as a sequence of blocks, and stored blocks are indexed by the hash of their contents. This solution, appropriate for immutable files, does not appear to be easily transposable to a setting with a very mutable content, in particular key indexing.

The same hot-spot phenomena that is found on the words that describe published contents can also be found on the words that are used for searching. Reynolds and Vahdat [13] have addressed the problem of query hot-spots and shown that single words on queries also follow a Zipf distribution. One should note, however, that the most popular words on the descriptions are not necessarily the most popular ones on queries. As a consequence, content hot-spots created by insert requests may not match temporal hot-spots resulting from popular queries.

Query hot-spots have a temporal nature and can be addressed by caching strategies. However, when coinciding with key-image hot-spots they can exacerbate the problem by adding more communication load to the already existing space and communication burden.

In order to address scalability in the co-domain it seems necessary to dimension resources by striving for a balanced distribution of the key-image load. Thus, abandoning the assumption that balancing the distribution of keys is, by itself, sufficient.

3 Bubble Grouping

Our design, of what we will refer to as *bubble grouping*, builds on the requirement that the indexing load should not surpass a given limit for each participating node, even when indexing a very popular key. Consequently, a very popular key might be indexed along several nodes, and each must be contacted in order to re-construct the whole key-image.

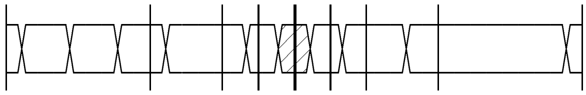

Figure 1: Any given bubble, here in gray, knows other bubbles at logarithmic distances on the key space.

Although a full description and analysis of this technique is out of the scope of this article we will concentrate on the main aspects of bubble grouping, in particular on those that depart from previous DHT designs.

In order to achieve resilience to node failures, nodes are organized into logical replication groups called *bubbles*. In each bubble all nodes replicate the same state, up to some appropriate consistency criteria. The number of nodes in each bubble should be kept in a soft variation band, with a lower limit r and an upper limit $2r$, and r should be big enough in order to tolerate high failure rates from the nodes associated to domestic users [10].

Bubbles form a logical ring on top of an overlay network, so that the key domain (m-bit keys resulting from the hash of single words) is divided along all the bubbles in the system. Each bubble will map a given key interval and be responsible for storing the keys, and associated key-images, that fit that interval. With popular keys one might find that some bubbles are only required to store a single key, and possibly only a portion of the key-image.

After initially suspecting that the routing solutions that assumed domain balancing would not be adaptable to the new setting, we have found that the Chord [16] philosophy could be transformed in order to provide the same scalability properties in a co-domain balanced setting. This will ensure that routing tables are $O(\log k)$ and that key placeholders are found with $O(\log k)$ jumps, where k is the magnitude of the key domain[1].

Bubbles in the ring are aware of a small set ($O(\log k)$) of other bubbles that are logarithmically spaced from its location in the ring, as depicted in Figure 1. The closest bubbles in this set will be the two neighbor bubbles.

The balancing of the storage state can be achieved by a local policy that, when thresholds are reached, distributes contents to the vicinity bubbles. When this occurs some state is migrated to adjacent bubbles and the key intervals of those bubbles are adjusted. As the system evolves, in

[1] To avoid collisions the size of the key domain, $k = 2^m$, must be driven by the vocabulary size, which in turn grows sub-linearly with the number of indexed contents. Thus, for large network sizes n, one can see that k will be less than n.

run-time, bubbles can travel to adjacent zones of the key space (something which is not possible on most DHT designs). Gradually, local hot-spots can be smoothened and the system will progressively converge to a globally balanced distribution of the mapping state.

When a new node joins the system by contacting any bubble it can be directed to the bubble in the vicinity that holds fewer nodes. Finally, when bubbles go bellow r they should join with the smallest adjacent bubble in order to maintain the intended degree of fault tolerance. Conversely a bubble exceeding $2r$ can be partitioned into two consecutive bubbles and generate more storage space for keys.

4 Content Based Indexing

Search engines are now a basic instrument for efficient use of the information that is provided on the Web. With P2P content sharing systems the search engine cataloging philosophy is not applicable, since node life time[2] is too small to allow an effective crawl on the system [10, 15]. Consequently the alternative would be global query dissemination (which is unlikely to become scalable) or content announcement and P2P indexing.

While not developing the issue of content ranking, the discussion bellow will try to give some insight on the issue of content announcement and querying, and propose a solution philosophy that keeps the approach scalable.

4.1 Measuring the Index

When addressing content based indexing for text objects one should be aware that for each quantity of indexed documents, that are announced to the DHT, a given load of indexing can be expected to be distributed throughout the system. It is relevant to inquire what proportion of indexing is generated for a given quantity of indexed text.

In order to access this we proceed with a simplistic analysis of the expected impact for each MB of new text data that is inserted into a running system. Since we are not doing a full analysis, we will assume that the impact of new words is neglectable (considering a stable running system) and do not consider compressed representations of the co-domain (since text is even more compressible than indexes). In fact, by neglecting those factors the numbers will be more conservative than the real ones.

The number of unique words per (1) MB of text can be approached by considering the average word length (in English) as 5 characters [2], calculating the expected number of words by $\frac{2^{20}}{5+1} \approx 175000$, and determining the expected number of unique words under the Heaps' Law[3] [8, 2], $30\sqrt{175000} \approx 12550$. Each unique word has an indexing impact of 8 bytes (IP, port and local 16 bit offset) leading to an impact of 10400 bytes (around 10% of 1 MB).

These estimates are comparable to the analysis of inverted index sizes in centralized indexing. In [2] the figures, for 10KB documents in collections from 1MB to 2GB of text, indicate impacts that range from 26% to 47% (and 19% to 26% when removing stopwords).

Since the replication factor, in P2P settings encompassing domestic users, might have to be around 20 [10] (since node failure rate is extremely high in their first hour of uptime), it is easy to conclude that the payload, which is at best a sizable fraction of the indexed text, will usually match its size or even multiply it by a small factor.

4.2 Scaling Down

Under the present trade-offs between disk storage and memory costs, it can still be argued that the expected index payload is undesirable but tolerable. It does not, by itself, hamper scalability but it does have a non neglectable impact on the overall communication load, since it will grow linearly with the total text object state.

It is known, from research on text content indexing [2], that not all words have equal importance on index construction for future querying. Here, we will concentrate on the fact that words that are present on a large proportion of documents are not by itself of particular relevance for direct indexing and querying.

A search for `mp3` that returns five thousand results does not convey significant information to the user if it cannot manage more than a small set of screen listings and a ranking mechanism is not present.

Even with a ranking mechanism, the user can only process a given number of results, leading to the intuition that an upper bound on the key-image size can be applied if appropriate compensation mechanisms are put to use.

One such mechanism is to forward a multiple word query only to the hosts that are associated to the less common keys (in the query). The results, arriving at the querier, should be few because popular key indexer hosts were excluded on the forwarding. With this list, the querier can ask each potential host to locally check for conformance with the other keywords (or even avoid this step if an OR search is the objective, since those are potentially the most relevant results).

This class of mechanisms can be combined with a scalable way of reducing key-image growth for very popular words. Those are the words in the first hundred positions of the Zipf distribution [6, 2].

[2] When considering domestic users.

[3] Heaps' Law [8, 2] is an empirical formula that calculates the expected number of unique words V (vocabulary) in a text as a function of the total number of words n under two constants, K and β. $V = Kn^{\beta}$, with $\beta \approx 0.4 - 0.6$ and $K \approx 10 - 100$. Here, as in [2], we consider $\beta = \frac{1}{2}, K = 30$.

4.3 Scalable Insertion Filtering

In order to establish an upper bound to the key-image it is not sufficient to simply enforce a local limit for each key – such limit will ensure that space bounds can be kept without resorting to multiple bubbles storing a single key. However, since filtering would be applied at the end of the routing process, an excessive communication load would still be observed on those hosts, on their vicinity and along the overall network.

It is better to filter the unwanted insertions along the routing path, ensuring that the globally popular keywords are filtered on the first hosts in the path. Each host in the insertion path would apply its filter and reduce the traffic for common keywords. Ideally the cumulative effect of filters should be such that the rate of arrival of insertions for all keys (popular or not) at the bubble that stores it, does not exceed a given value.

In order to avoid discarding significant documents for a given common keyword, filtering should take into account the local frequency of each keyword in the document. Keywords with high relative frequency in the associated document, should have lower probabilities of seeing their insert request discarded by the filtering mechanism.

Under filtering it would be easy to apply a local rejection policy that deletes older entries in the key-image enforcing some local limit. This can be complemented by a re-announcement policy, calibrated according to the node life time distribution, where each node periodically (re-)publishes its contents. Such strategy provides an easy approach to the management of stale entries in the distributed index.

4.4 Exploratory Simulation

To probe the feasibility of this design we conducted a simulation of a bubble grouping DHT where each bubble is assigned a range of the key domain that is calibrated under the expected load in the key-image. This load is known in advance since each key is hashed from an English vocabulary where each key is associated to its probability of occurrence. A real bubble grouping system would be expected to evolve, under local load balancing, into this state.

Here, the number of nodes, the mapped key range and the routing tables are all fixed in advance. The simulation is then conducted by generating a number of key insert requests, where each request is originated in a random node/bubble. The published keys are randomly chosen according to their frequencies[4].

If no insertion filters are applied all the key insert requests will reach their destinations after a small set of forwarding jumps. The lines on Figures 2 and 3 show the number of insert requests for each key, under decreasing order on the number of insert requests. As expected, they follow a Zipf distribution and in a logarithmic scale the distribution is almost linear.

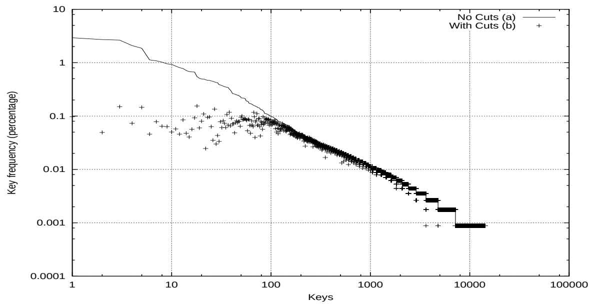

Figure 2: Insertion reduction with an I_4 filter.

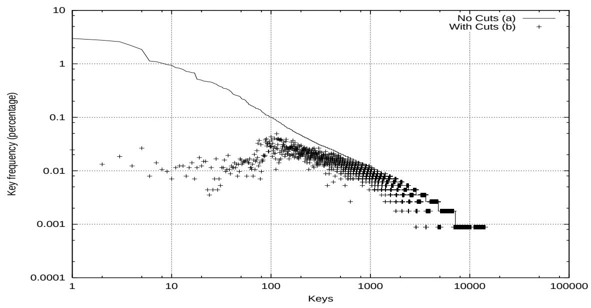

Figure 3: Insertion reduction with an I_{32} filter.

Node filters should be able to reduce the number of insert requests that reach their destination when the keys are popular (left hand side of the figures) and should not have a significant influence on those associated to less frequent keys (right hand side). These filters will have no knowledge of the global key frequencies and can only act with local information.

Our basic filter design gathers information on the last K insert requests, on a set I_K. Each node can determine if a given insert request i belongs or not to I_K. In the simulation I_K is implemented by Bloom Filters [3].

In the experiment we compare two filter alternatives when deciding if an insert is to be dropped: $i \in I_4$; $i \in I_{32}$. Figures 2 and 3 compare the number of inserts after applying one of these two filters against the reference line of unfiltered inserts.

The I_{32} filter, Figure 3, provides a radical reduction on the common keys. However this is achieved with the cost of introducing some perturbation on less common keys. In contrast, the I_4 filter, Figure 2, is almost innocuous to the less common keys, while still having a considerable effect on the reduction of common keys.

As desired both filters have most impact on the first 100 of the generated 14000 distinct keys ($\approx$ 1%). This is basically the share of keys who is responsible for key-image hot-spots, and both filters manage to keep their impacts at rates that are significantly lower. This is, respectively, around 0.1%

[4]For space saving reasons the local keyword document frequency is not considered in this simulation.

and 0.01% for I_4 and I_{32}, whilst the unfiltered line shows impacts above 1% for the initial keys. Actually, with no filters, the most popular key has 6.4% impact, which means that a host that is associated to that key would have to cope with this percentage of the whole distributed insertion load in the system.

This experiment provides the basic insight into the potential of insertion filtering for key-image hot-spot contention. In order to obtain the full functionality one would have to ensure that: popular content keys are detected when forwarding queries, confining query forwarding to the less common keys; common keys are dropped as soon as possible in the route; filters take into account local keyword document frequencies, in order to drop the less significant associations.

5 Conclusions

The P2P concept can be applied to very distinct scenarios. On one extreme it can provide a new design philosophy to build distributed applications running on groups of well administrated servers that provide a defined functionality to a specific set of users. At the other end, one can find the, by far, most popular application of P2P: content sharing among end users. In a way, this mimics what happened in the initial years of the WWW, when end users started to publish their own content and pages.

When considering P2P with a major share of end users, one must accept that the average node life time will be extremely low, typically bellow one hour. There are two major consequences: the search engine approach will not be adequate; algorithms that try to tame query broadcasts by passively learning about content placement will likely fail to adapt to very dynamic networks. The remaining alternative is content announcement and DHT algorithms.

We have argued that present DHT algorithms do not adapt, in terms of load scalability, to the mapping of keywords that follow uneven distributions (with key-image hot-spots). This is the case of some popular file descriptions, such as `mp3`, as also is the case in the Zipf distribution that characterizes text based contents.

We have shown that these limitations can be addressed by a DHT design that distributes load according to the size of the key-image of indexed keys. This design keeps the usual scalability properties of previous DHT algorithms.

Additionally, we have argued that the extra load that comes from migration from description based indexing to content based indexing requires a distributed mechanism that minimizes the insertion of very common keys. After sketching a search mechanism that compensates the missing insertions we concluded by testing the feasibility of distributed filtering mechanisms.

By considering the impact that search catalogs had on the Web, we believe that content search is a crucial factor and that it can apply a major modification to the way in which P2P systems are currently used.

References

[1] Lada Adamic, Rajan Lukose, Amit Puniyani, and Bernardo Huberman. Search in power-law networks. *Physical Review E*, 64(046135), 2001.

[2] R. Baeza-Yates and Ribeiro-Neto B. *Modern Information Retrieval.* ACM Press, NewYork, 1999.

[3] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *International Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, Berkeley, CA, 2000.

[5] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*, Banff, Canada, October 2001.

[6] Zipf G. *Human Behaviour and the Principle of Least Effort.* Addison-Wesley, 1949.

[7] Gnutella website. http://gnutella.wego.com/.

[8] Heaps J. *Information Retrieval – Computational and Theorectical Aspects.* Academic Press, 1978.

[9] Amr Z. Kronfol. *FASD: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine.* PhD thesis, May 2002.

[10] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, Cambridge, USA, March 2002.

[11] Napster. http://www.napster.com.

[12] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. In *Proceedings of the ACM SIGCOMM'01 Conference*, pages 161–172, 2001.

[13] P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. http://issg.cs.duke.edu/search/. Unpublished manuscript.

[14] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms*, Germany, 2001.

[15] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, CA, USA, January 2002.

[16] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the ACM SIGCOMM'01 Conference*, pages 149–160, 2001.

[17] Beverly Yang and Hector Garcia-Molina. Efficient search in peer-to-peer networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–14. IEEE, 2002.

[18] B. Zhao, J. Kubiatowicz, and A. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, 2001.